Operating System - Course Project Specifications

Platforms & Languages

- **Languages:** Java (preferred), C++, C#, VB.NET, VC++.NET
- **Platforms:** Windows or Linux

Project Groups

2 to 3 members

Coding Style

Make sure your program is commented properly. No matter how 'obvious' (,'clear', 'self explanatory', etc. etc. etc.) variable and function names are, you MUST add comments in all your source code files. Object Oriented Programming should be used. Proper coding conventions must be followed whatever language you choose.

Cheating

The **maximum** penalty for plagiarism of any kind will be a **Disciplinary Committee call** and **minimum** will be the **-100 policy**.

Phase I

1. VM Architecture

What to do?

- Build main architecture
- Implement instruction set

Main Memory

- Memory addressing is 16 bits
- Memory is simulated by taking a byte (unsigned char) array of 64K
- All memory references are through Load and Store instructions between Memory and General purpose Registers
- Stack is of ***60 bytes*** for each process

Registers

- The architecture has sixteen, 16-bit general purpose registers
- Sixteen, 16-bit special purpose registers:
  - The value of first register is always Zero
  - Three registers for code (base, limit, & counter)
  - Three registers for stack (base, limit, & counter)
  - Two registers for data (base & limit)
  - One for flags
  - Six Registers for reserved future use
- Register code is of one byte
- The register codes are as follows:

Register	Register Code
R0	00h
R1	01h
R2	02h
…	…
R10	0Ah
…	…
R14	0Eh
R15	0Fh

- Use the flag register as follows:

Use	Unused		Overflow	Sign	Zero	Carry
Bit No.	…	4	3	2	1	0

- **Flag Register** will be set after Arithmetic, logical, Shift and Rotate Operations as follows:
  - **Carry Bit:** Set for **Shift and Rotate** operations only. If the most significant bit is on before the operation.
  - **Zero Bit:** If the result of **Arithmetic, logical, Shift and Rotate o**perations is Zero.
  - **Sign Bit:** If the result of **Arithmetic, logical, Shift and Rotate o**perations is Negative.
  - **Overflow Bit:** If the result of **Arithmetic and logical o**perations is either carry in or carry out of the most significant bit.

Instruction Set

- All arithmetic is integer
- All numbers are signed
- Numbers are stored in memory in little-endian
- There is a single addressing mode, 16 bit signed offset
- All Instructions are of 2 clock cycles
- Details of memory addressing are found on memory management section
- All access violations (accessing out of bounds code, data, stack overflow & underflow) must be trapped and an error must be generated so that the Operating System component could act on it (kill the task after displaying error)

Opcode (hex)	Instruction	Description	Example	Details
Register-register Instructions				
14	MOV	Copy Register Contents	MOV R1 R2	R1 ← R2
18	ADD	Add Register Contents	ADD R1 R2	R1 ← R1 + R2
1A	SUB	Subtract	SUB R1 R2	R1 ← R1 – R2
1C	MUL	Multiply	MUL R1 R2	R1 ← R1 * R2
1E	DIV	Division	DIV R1 R2	R1 ← R1 / R2
20	AND	Logical AND	AND R1 R2	R1 ← R1 && R2
22	OR	Logical OR	OR R1 R2	R1 ← R1 || R2
Register-Immediate Instructions				
15	MOVI	Copy Immediate to register	MOVI R1 num	R1 ← num
19	ADDI	Add	ADDI R1 num	R1 ← R1 + num
1B	SUBI	Subtract	SUBI R1 num	R1 ← R1 - num
1D	MULI	Multiply	MULI R1 num	R1 ← R1 * num
1F	DIVI	Divide	DIVI R1 num	R1 ← R1 / num
21	ANDI	Logical AND	ANDI R1 num	R1 ← R1 && num
23	ORI	Logical OR	ORI R1 num	R1 ← R1|| num
32	BZ	Branch equal to zero	BZ num	Check flag register, and jump to offset **
33	BNZ	Branch if not zero	BNZ num	Check flag register, and jump to offset **
34	BC	Branch if carry	BC num	Check flag register, and jump to offset **
35	BS	Branch if sign	BS num	Check flag register, and jump to offset **
36	JMP	Jump	JMP num	Jump to offset **
40	CALL	Procedure Call	CALL num	Push PC on stack, Jump to offset **
*51	ACT	Action	ACT num	Do the service defined by num
Memory Instructions				
*16	MOVL	Load Word	mov R1 offset	R1 ← Mem [location**]
*17	MOVS	Store Word	mov R1 offset	Mem [location**] ← R1
Single Operand Instructions				
24	SHL	Shift Left Logical	SHL R1	R1 ← R1 << 1
25	SHR	Shift Right Logical	SHR R1	R1 ← R1 >> 1
26	RTL	Rotate Left	RTL R1	Shift left and set lower bit accordingly
27	RTR	Rotate Right	RTR R1	Shift right and set lower bit accordingly
28	INC	Increment	INC R1	R1 ← R1 + 1
29	DEC	Decrement	DEC R1	R1 ← R1 – 1
2A	PUSH	Push register on stack	PUSH R1	Push contents of R1 on stack
2B	POP	Pop the value in the register from the stack	POP R1	Pop contents of top of stack on R1
No Operand Instructions				
41	RETURN	Return to original PC	RETURN	Pop PC from Stack
2F	NOOP	No Operation	NOOP	No Operation
FF	END	End of Process	END	Process Terminates

* Implementation of these instructions might have to be modified in the next parts

** Memory references can either be absolute address in the 64K memory, or could be a offset which should be used to compute the absolute address according to the technique described in memory management section. In case you decide to use absolute addressing, then your OS will do the address translation.

Instruction Format

- Instruction op-code is of 1 byte
- Register code is 1 byte
- Immediate = 2 bytes

Register-Register Instructions

Size = 3 bytes

Op-code	Register1	Register2
8 bits	8 bits	8 bits

Usage:

- **Register-Register ALU Operations**: Register1 ← Register1 op Register2

Register-Immediate Instructions

Size = 4 bytes

Op-code	Register	Immediate
8 bits	8 bits	16 bits

Usage:

- **Register-Immediate ALU Operations:** Register ← Register op Immediate
- **Call / Jump Instructions:** PC = code.Base + num (after necessary checks) Note that Register is not used here

Memory Instructions

Size = 4 bytes

Op-code	Register	Immediate
8 bits	8 bits	16 bits

Usage:

- **Storing Value in Memory:** memory [data.base + imm] ← Register
- **Loading Value in Memory:** Register ← memory [data.base + imm]

Single-Operand Instructions

Size = 2 bytes

Op-code	Register
8 bits	8 bits

Usage:

- **Push / Pop Instructions:** push or pop the register contents to or from stack
- **Other Instructions:** R1 ← R1 op

No-Operand Instructions

Size = 1 bytes

Op-code
8 bits

Machine execution cycle

Each cycle contains the following steps:

- Fetch the instruction
- Decode the fetched instruction
- Execute the instruction.
- Write back result in the memory (if necessary)